**Neural Networks.**
**Lab 2: Single layer perceptrons. Linearly separable classification problems.**

---

## 1. Defining single layer perceptrons.

A single layer perceptron consists of a layer functional units having binary activation functions (e.g. Heaviside (Matlab function: hardlim) and signum (Matlab function: hardlims)).

If the patterns to be classified are N-dimensional then the networks should have N input units. If the input patterns have to be classified in 2 classes then one can use just one functional unit and interpret its values as follows: if the output value is 0 (or -1) then the input pattern belongs to class 1 otherwise it belongs to class 2.

If the patterns should be classified in M classes the usual approach is to use M binary functional units and interpret as resulting class the index of the unit which generates a value equal to 1. If the output vector contains several values equal to 1 this means that the network cannot find the right class corresponding to the input pattern.

The function used to define a single layer perceptron is newp. In the case of a network with N input units and M functional units its syntax is:

nnObject = newp([min1 max1; ...; minN maxN], M, '<activation function>', '<learning algorithm>')

The parameters involved in the newp call are:

[min1 max1; ...; minN maxN]: a matrix with N rows containing for each input unit the limits of the range of corresponding input values. If the input vector is a binary one then for each index i, the lower bound will be mini=0 and the upper bound will be maxi=1.

M: the number of output units

'<activation function>': optional parameter specifying the name of the activation function. The possible values are: 'hardlim' (implicit value when the parameter is missing) and 'hardlims'.

'<learning algorithm>': optional parameter specifying the name of the learning algorithm. The possible values are: 'learnp' (classical algorithm for perceptron training - implicit value when the parameter is missing) and 'learnpn' (normalized version of the perceptron's algorithm: the vectors containing the weights are normalized during the learning process.

Remark 1: newp creates an neural network – type object consisting of several fields. In order to visualize the values of the synaptic weights it is enough to specify: nnObject.IW and in order to visualize the biases associated to the functional units it is enough to specify: nnObject.b

Remark 2: there is another way to call newp: by specifying as first parameter the inputs in the training set and as second parameter the corresponding desired outputs, e.g. newp(inputPatterns, desiredOutputs)

## 2. Training single layer perceptrons.

The training process is based on a training set and a training algorithm. The training set consists of the input patterns and the desired outputs. Each input pattern should be specified as a column vector, thus a set of L input patterns of size N can be represented as a matrix having N rows and L columns. In the case of a training set containing L examples the desired outputs should be represented as a row vector containing L elements.

The process of training a network nnObject can be realized by using the function adapt (which implements the perceptron's learning algorithm). The syntax of this function is:

```
nnTrained=adapt(nnObject, inputPatterns, desiredOutputs)
```

In the implicit variant the training set is scanned just once. Sometimes just once pass through the training set is not enough, thus we have to specify the number of passes by setting the corresponding parameter of the adapt function, e.g.:

```
nnObject.adaptParam.passes=10
```

## 3. Simulating single layer perceptrons.

In order to compute the output of a network one can use the function sim, having the syntax:

```
sim(nnTrained, testPatterns)
```

## 4. Applications.

### 4.1. Representation of Boolean functions.

To represent the Boolean function corresponding to disjunction (OR) one can use a perceptron containing two input units and one output unit.

```
percOR=newp([0 1; 0 1],1);                % perceptron definition
input = [0 0 1 1; 0 1 0 1]; d=[0 1 1 1];  % training set
plotpv(input,d);                          % visualizing the data in the training set
percOR.adaptParam.passes=20;              % set the number of passes
percORa=adapt(percOR,input,d);            % network's training
plotpc(percORa.IW{1},percORa.b{1});       % visualized the decision line
rez=sim(percORa,[0 0 1 1; 0 1 0 1])       % testing the network
```

Exercise: 1. train the same perceptron to represent the conjunction (AND)
2. solve the same problem by using the GUI: nntool (hint: first you should define the input and output data and select the perceptron from the list of implemented neural networks).

### 4.2. Classification of patterns specified as binary matrices

Let us consider a set of pattern specified as matrices containing the value 1 where a pixel is active and 0 where a pixel is inactive. A function for reading such patterns is described in the following. It should be called by specifying the size of the pattern: the number of rows followed by the number of columns.

```
function [ss]=readPattern(m,n)
% m  = number of rows
% n = number of columns
clf                                 % clean the graphic windows
axis square
axis([0 n 0 m])                     % define the coordinates system
hold on
for i=0:m                           % draw the grid
  line([0,n],[i i]);
end
for j=0:n
  line([j,j], [0 m]);
end
s=zeros(m,n);                       % initialize the matrix
b = 1;
while b == 1
 [xi,yi,b] = ginput(1);              % analyze the event produced by a mouse click
 if b==1                            % left click
   i=floor(yi)+1; j=floor(xi)+1;    % identify the cell containing the mouse arrow
   s(m-i+1,j)=1-s(m-i+1,j);         % change the state of the cell
   if s(m-i+1,j)==1
        patch([j-1 j j j-1],[i i i-1 i-1],'r');   % red: active cell
   else
        patch([j-1 j j j-1],[i i i-1 i-1],'w');  % white: inactive cell
   end
 end
end
for i=1:m
  ss((i-1)*n+1:i*n)=s(i,:)';        % linearized version of the matrix s
end
end
```

Exercise: Fill in a few patterns and analyze the content of the generated matrix.

A perceptron able to classify some input patterns in some predefined classes can be designed as follows:

```
function [s,d]=classification(m,n,L,M)
% m = number of rows of each pattern
% n = number of columns of each pattern
% L = number of examples in the training set
% M = number of classes
d=zeros(M,L);                    % initialize the training set
s=zeros(m*n,L);
for i=1:L                        % reading the training set
   s(:,i)=readPattern(m,n);      % input pattern
   r=input('Correct class:');    % desired output (index of the correct class)
   d(r,i)=1;                     % construction of the correct output vector
end
perc=newp(s,d);                  % create the perceptron
perc.adaptParam.passes=20;       %  set the number of passes
perca=adapt(perc,s,d);           % train the network
disp('Test pattern:');
t=readPattern(m,n);
rez=sim(perca,t');
disp('Answer:'); disp(rez);
end
```

Exercise: train the network to classify simple shapes (horizontal, vertical and diagonal lines) represented as small images (e.g. 3*3) and test it for a slightly altered shape (e.g. one cell is complemented)